```
import React, { useState, useEffect, useRef } from 'react';
import { initializeApp } from 'firebase/app';
import { getAuth, signInAnonymously, signInWithCustomToken,
onAuthStateChanged } from 'firebase/auth';
import { getFirestore, doc, setDoc, onSnapshot, collection, query,
getDocs } from 'firebase/firestore';
import { Eye, Pencil, RotateCcw, RotateCw } from 'lucide-react'; //
Importing icons

// Define global variables provided by the Canvas environment
const appId = typeof __app_id !== 'undefined' ? __app_id : 'default-
app-id';
const firebaseConfig = typeof __firebase_config !== 'undefined' ?
JSON.parse(__firebase_config) : {};
const initialAuthToken = typeof __initial_auth_token !== 'undefined' ?
__initial_auth_token : null;

// Initialize Firebase outside the component to avoid re-
initialization
let app;
let db;
let auth;

try {
  app = initializeApp(firebaseConfig);
  db = getFirestore(app);
  auth = getAuth(app);
} catch (error) {
  console.error("Firebase initialization error:", error);
}

// Helper to generate a unique ID for new cultivators if needed
const generateUniqueId = () => Math.random().toString(36).substring(2,
9);

// Predefined list of cultivators for the palette with assigned colors
(updated from provided images)
const initialCultivators = [
  { id: generateUniqueId(), name: 'AERIZ', color: 'bg-green-500' },
  { id: generateUniqueId(), name: 'ASCEND', color: 'bg-blue-500' },
  { id: generateUniqueId(), name: 'BEDFORD GROW', color: 'bg-pink-300'
},
  { id: generateUniqueId(), name: 'CNS HARVEST', color: 'bg-red-600'
},
  { id: generateUniqueId(), name: 'CONTRACT CANNA', color: 'bg-stone-
500' },
  { id: generateUniqueId(), name: 'CRESCO', color: 'bg-gray-500' },
  { id: generateUniqueId(), name: 'ELEVATED ORGANIX', color: 'bg-blue-
700' },
  { id: generateUniqueId(), name: 'FLORA ARBOR', color: 'bg-blue-900'
},
  { id: generateUniqueId(), name: 'KROWN', color: 'bg-fuchsia-800' },
```

```
  { id: generateUniqueId(), name: 'NATURE\'S GRACE & WELLNESS', color:
'bg-emerald-300' },
  { id: generateUniqueId(), name: 'REDEMPTION', color: 'bg-blue-700'
},
  { id: generateUniqueId(), name: 'REVOLUTION', color: 'bg-purple-700'
},
  { id: generateUniqueId(), name: 'GRASSROOTS', color: 'bg-green-600'
},
  { id: generateUniqueId(), name: 'GREEN GROWTH GROUP', color: 'bg-
green-800' },
  { id: generateUniqueId(), name: 'GTI', color: 'bg-purple-600' },
  { id: generateUniqueId(), name: 'HDC GROUP', color: 'bg-orange-400'
},
  { id: generateUniqueId(), name: 'VERANO', color: 'bg-pink-500' },
  { id: generateUniqueId(), name: 'IESO', color: 'bg-yellow-400' },
  { id: generateUniqueId(), name: 'INGROWN FARMS (IGF)', color: 'bg-
teal-600' },
  { id: generateUniqueId(), name: 'KAVIAR', color: 'bg-indigo-900' },
];

// Cart component
const Cart = ({ cart, onDropCultivator, isDraggingCultivator, userId,
lastMovedInfo, onClearCart, isEditMode }) => {
  const cartRef = useRef(null);

  // Determine if a slot is valid for dropping (only in edit mode)
  const handleDragOver = (e) => {
    e.preventDefault(); // Necessary to allow dropping
    if (isEditMode) {
      e.dataTransfer.dropEffect = 'move';
    }
  };

  // Handle dropping a cultivator onto a slot (only in edit mode)
  const handleDrop = (e, sideIndex, slotIndex) => {
    if (isEditMode) {
      e.preventDefault();
      const cultivatorData = e.dataTransfer.getData('cultivator');
      if (cultivatorData) {
        const { id, name, color } = JSON.parse(cultivatorData);
        // Pass the new cultivator to the parent to add it to the
array
        onDropCultivator(cart.id, sideIndex, slotIndex, { id, name,
color }, 'add');
      }
    }
  };

  // Handle clicking a slot to clear the last cultivator (only in edit
mode)
  const handleSlotClick = (sideIndex, slotIndex) => {
    if (isEditMode) {
```

```
      // Only clear if there are cultivators in the slot
      if (cart.sides[sideIndex].slots[slotIndex].cultivators.length >
0) {
        // Pass null and 'remove' action to remove the last one
        onDropCultivator(cart.id, sideIndex, slotIndex, null,
'remove');
      }
    }
  };

  return (
    <div
      ref={cartRef}
      className="bg-gray-700 p-2 rounded-lg shadow-lg flex flex-col
items-center justify-between w-full h-60 md:h-80 relative"
    >
      <h3 className="text-white text-lg font-semibold mb-
2">{cart.name}</h3>
      <div className="flex flex-1 w-full relative"> {/* flex-1 to take
available height, relative for absolute positioning */}
        {/* Left Side */}
        <div className="flex flex-col justify-around w-1/2 p-1 border-
r border-gray-600">
          <p className="text-gray-400 text-xs mb-1 text-center font-
bold">{cart.sides[0].name}</p>
          {cart.sides[0].slots.map((slot, slotIndex) => (
            <div
              key={slotIndex}
              className={`flex-1 flex flex-wrap items-center justify-
center p-1 m-1 border-2 border-dashed rounded-md text-sm text-gray-300
relative gap-1
              ${isEditMode ? 'cursor-pointer' : 'cursor-default'}
              ${isDraggingCultivator ? 'border-blue-400' :
(slot.cultivators.length > 0 ? slot.cultivators[0].color : 'border-
gray-500')}`}
              onDragOver={handleDragOver}
              onDrop={(e) => handleDrop(e, 0, slotIndex)}
              onClick={() => handleSlotClick(0, slotIndex)}
              title={
                slot.cultivators.length > 0
                  ? slot.cultivators.map(c => `Cult.:
${c.name}\nLevel: ${slot.level}\nLast Moved By:
${lastMovedInfo[c.id]?.by || 'N/A'}\nLast Moved At:
${lastMovedInfo[c.id]?.at || 'N/A'}`).join('\n---\n')
                  : `Drag Cult. here or click to clear last (Level
${slot.level})`
              }
            >
              {slot.cultivators.length > 0 ? (
                slot.cultivators.map((cultivator, cultIndex) => (
                  <div key={cultIndex} className={`${cultivator.color}
text-white font-medium text-center px-1 py-0.5 rounded-sm text-xs`}>
```

```
                    {cultivator.name}
                    <span className="block text-xs text-gray-200">Lvl:
{slot.level}</span>
                  </div>
                ))
              ) : (
                <span className="text-gray-400">Lvl:
{slot.level}</span>
              )}
            </div>
          ))}
        </div>

        {/* Central Arrows */}
        <div className="absolute inset-y-0 left-1/2 -translate-x-1/2
flex flex-col justify-between items-center py-4">
          {/* Top Arrow */}
          <svg width="20" height="20" viewBox="0 0 24 24" fill="none"
stroke="currentColor" strokeWidth="2" strokeLinecap="round"
strokeLinejoin="round" className="text-gray-400">
            <path d="M12 19V5M5 12l7-7 7 7"/>
          </svg>
          {/* Bottom Arrow */}
          <svg width="20" height="20" viewBox="0 0 24 24" fill="none"
stroke="currentColor" strokeWidth="2" strokeLinecap="round"
strokeLinejoin="round" className="text-gray-400">
            <path d="M12 5v14M19 12l-7 7-7-7"/>
          </svg>
        </div>

        {/* Right Side */}
        <div className="flex flex-col justify-around w-1/2 p-1 border-
l border-gray-600">
          <p className="text-gray-400 text-xs mb-1 text-center font-
bold">{cart.sides[1].name}</p>
          {cart.sides[1].slots.map((slot, slotIndex) => (
            <div
              key={slotIndex}
              className={`flex-1 flex flex-wrap items-center justify-
center p-1 m-1 border-2 border-dashed rounded-md text-sm text-gray-300
relative gap-1
              ${isEditMode ? 'cursor-pointer' : 'cursor-default'}
              ${isDraggingCultivator ? 'border-blue-400' :
(slot.cultivators.length > 0 ? slot.cultivators[0].color : 'border-
gray-500')}`}
              onDragOver={handleDragOver}
              onDrop={(e) => handleDrop(e, 1, slotIndex)}
              onClick={() => handleSlotClick(1, slotIndex)}
              title={
                slot.cultivators.length > 0
                  ? slot.cultivators.map(c => `Cult.:
${c.name}\nLevel: ${slot.level}\nLast Moved By:
```

```
${lastMovedInfo[c.id]?.by || 'N/A'}\nLast Moved At:
${lastMovedInfo[c.id]?.at || 'N/A'}`).join('\n---\n')
                  : `Drag Cult. here or click to clear last (Level
${slot.level})`
              }
            >
              {slot.cultivators.length > 0 ? (
                slot.cultivators.map((cultivator, cultIndex) => (
                  <div key={cultIndex} className={`${cultivator.color}
text-white font-medium text-center px-1 py-0.5 rounded-sm text-xs`}>
                    {cultivator.name}
                    <span className="block text-xs text-gray-200">Lvl:
{slot.level}</span>
                  </div>
                ))
              ) : (
                <span className="text-gray-400">Lvl:
{slot.level}</span>
              )}
            </div>
          ))}
        </div>
      </div>

      {/* "X" button to clear the cart - only visible in edit mode */}
      {isEditMode && (
        <button
          onClick={() => onClearCart(cart.id)}
          className="absolute top-2 right-2 bg-red-700 hover:bg-red-
800 text-white text-xs font-bold w-6 h-6 flex items-center justify-
center rounded-full shadow-md"
          title="Clear all cultivators from this cart"
        >
          X
        </button>
      )}
    </div>
  );
};

// Main App component
const App = () => {
  const [carts, setCarts] = useState([]);
  const [userId, setUserId] = useState(null);
  const [isAuthReady, setIsAuthReady] = useState(false);
  const [isDraggingCultivator, setIsDraggingCultivator] =
useState(false);
  const [savingStatus, setSavingStatus] = useState('');
  const [lastMovedInfo, setLastMovedInfo] = useState({}); // Stores {
cultivatorId: { by: userId, at: timestamp }}

  // New states for Undo/Redo and Edit/View mode
```

```
  const [history, setHistory] = useState([]);
  const [historyIndex, setHistoryIndex] = useState(-1);
  const [isEditMode, setIsEditMode] = useState(false); // Default to
view-only

  const stockroomRef = useRef(null);
  const stockroomLayoutDocRef = useRef(null);
  const isHistoryInitialized = useRef(false); // New ref to track
history initialization

  // Initialize Firebase and set up auth listener
  useEffect(() => {
    if (!app || !db || !auth) {
      console.error("Firebase not initialized.");
      return;
    }

    const unsubscribe = onAuthStateChanged(auth, async (user) => {
      if (user) {
        setUserId(user.uid);
        console.log("User authenticated:", user.uid);
      } else {
        try {
          if (initialAuthToken) {
            await signInWithCustomToken(auth, initialAuthToken);
          } else {
            await signInAnonymously(auth);
          }
          setUserId(auth.currentUser?.uid || crypto.randomUUID());
          console.log("Signed in anonymously or with custom token.");
        } catch (error) {
          console.error("Firebase authentication error:", error);
          setUserId(crypto.randomUUID());
        }
      }
      setIsAuthReady(true);
    });

    return () => unsubscribe();
  }, []);

  // Fetch and listen to stockroom layout data
  useEffect(() => {
    if (!isAuthReady || !db || !userId) {
      return;
    }

    stockroomLayoutDocRef.current = doc(db,
`artifacts/${appId}/public/data/stockroomLayouts`, 'mainLayout');

    const unsubscribe = onSnapshot(stockroomLayoutDocRef.current,
      (docSnap) => {
```

```
        const data = docSnap.data();
        const loadedCarts = data?.carts || [];
        const loadedLastMovedInfo = data?.lastMovedInfo || {};

        setCarts(loadedCarts);
        setLastMovedInfo(loadedLastMovedInfo);

        // Only initialize history on the very first data load
        if (!isHistoryInitialized.current) {
          if (docSnap.exists()) {
            setHistory([{ carts: loadedCarts, lastMovedInfo:
loadedLastMovedInfo }]);
            setHistoryIndex(0);
            console.log("Stockroom data loaded and history initialized
from Firestore.");
          } else {
            console.log("No stockroom data found, initializing default
layout and history.");
            const defaultCarts = [
              { id: 'cart-1', name: 'Cart 1', sides: [{ name: 'Side
A', slots: Array.from({ length: 4 }, (_, i) => ({ cultivators: [],
level: i + 1 })) }, { name: 'Side B', slots: Array.from({ length: 4 },
(_, i) => ({ cultivators: [], level: i + 1 })) }] },
              { id: 'cart-2', name: 'Cart 2', sides: [{ name: 'Side
C', slots: Array.from({ length: 4 }, (_, i) => ({ cultivators: [],
level: i + 1 })) }, { name: 'Side D', slots: Array.from({ length: 4 },
(_, i) => ({ cultivators: [], level: i + 1 })) }] },
              { id: 'cart-3', name: 'Cart 3', sides: [{ name: 'Side
E', slots: Array.from({ length: 4 }, (_, i) => ({ cultivators: [],
level: i + 1 })) }, { name: 'Side F', slots: Array.from({ length: 4 },
(_, i) => ({ cultivators: [], level: i + 1 })) }] },
              { id: 'cart-4', name: 'Cart 4', sides: [{ name: 'Side
G', slots: Array.from({ length: 4 }, (_, i) => ({ cultivators: [],
level: i + 1 })) }, { name: 'Side H', slots: Array.from({ length: 4 },
(_, i) => ({ cultivators: [], level: i + 1 })) }] },
            ];
            setCarts(defaultCarts);
            setLastMovedInfo({});
            setHistory([{ carts: defaultCarts, lastMovedInfo: {} }]);
            setHistoryIndex(0);
            saveStockroomLayout(defaultCarts, {});
          }
          isHistoryInitialized.current = true; // Mark history as
initialized
        } else {
          console.log("Stockroom data updated from Firestore (not re-
initializing history).");
        }
      }, (error) => {
        console.error("Error listening to stockroom data:", error);
        setSavingStatus('Error loading data!');
      });
```

```
    return () => unsubscribe();
  }, [isAuthReady, userId]);

  // Function to save the current stockroom layout to Firestore
  const saveStockroomLayout = async (currentCarts,
currentLastMovedInfo) => {
    if (!stockroomLayoutDocRef.current || !userId) {
      console.warn("Firestore not ready or user not authenticated for
saving.");
      return;
    }
    setSavingStatus('Saving...');
    try {
      await setDoc(stockroomLayoutDocRef.current, {
        carts: currentCarts,
        lastMovedInfo: currentLastMovedInfo,
        updatedAt: new Date().toISOString(),
        updatedBy: userId,
      });
      setSavingStatus('Saved!');
      setTimeout(() => setSavingStatus(''), 2000);
    } catch (error) {
      console.error("Error saving stockroom layout:", error);
      setSavingStatus('Save failed!');
    }
  };

  // Function to add a state to history and save
  const addStateToHistoryAndSave = (newCarts, newLastMovedInfo) => {
    const newHistory = history.slice(0, historyIndex + 1); // Discard
future states if undo was used
    newHistory.push({ carts: newCarts, lastMovedInfo: newLastMovedInfo
});
    setHistory(newHistory);
    setHistoryIndex(newHistory.length - 1);
    saveStockroomLayout(newCarts, newLastMovedInfo);
  };

  // Handle drag start for cultivators (only in edit mode)
  const handleDragStartCultivator = (e, cultivator) => {
    if (isEditMode) {
      e.dataTransfer.setData('cultivator',
JSON.stringify(cultivator));
      setIsDraggingCultivator(true);
    } else {
      e.preventDefault(); // Prevent drag if not in edit mode
    }
  };

  // Handle drag end for cultivators
  const handleDragEndCultivator = () => {
```

```
    setIsDraggingCultivator(false);
  };

  // Handle dropping a cultivator onto a cart slot or removing one
  const handleDropCultivator = (cartId, sideIndex, slotIndex,
cultivatorData, action) => {
    const newCarts = carts.map(cart => {
      if (cart.id === cartId) {
        const newSides = [...cart.sides];
        const newSlots = [...newSides[sideIndex].slots];
        const currentCultivators =
[...newSlots[slotIndex].cultivators];
        let removedCultivator = null;

        if (action === 'add' && cultivatorData) {
          // Add the new cultivator to the array
          newSlots[slotIndex].cultivators = [...currentCultivators,
cultivatorData];
        } else if (action === 'remove') {
          // Remove the last cultivator from the array
          if (currentCultivators.length > 0) {
            removedCultivator = currentCultivators.pop(); // Remove
the last one
            newSlots[slotIndex].cultivators = currentCultivators;
          }
        }

        newSides[sideIndex] = { ...newSides[sideIndex], slots:
newSlots };
        return { ...cart, sides: newSides };
      }
      return cart;
    });

    setCarts(newCarts);

    const newLastMovedInfo = { ...lastMovedInfo }; // Create a copy
    if (action === 'add' && cultivatorData) {
      newLastMovedInfo[cultivatorData.id] = { by: userId, at: new
Date().toLocaleString() };
    } else if (action === 'remove' && removedCultivator) {
      delete newLastMovedInfo[removedCultivator.id];
    }

    setLastMovedInfo(newLastMovedInfo);
    addStateToHistoryAndSave(newCarts, newLastMovedInfo);
  };

  // Handle clearing all cultivators from a specific cart
  const handleClearCart = (cartId) => {
    const newCarts = carts.map(cart => {
      if (cart.id === cartId) {
```

```
        const clearedSides = cart.sides.map(side => {
          const clearedSlots = side.slots.map((slot) => {
            // Collect IDs of cultivators being cleared for
lastMovedInfo update
            slot.cultivators.forEach(cult => {
              if (lastMovedInfo[cult.id]) {
                delete lastMovedInfo[cult.id]; // Remove from
lastMovedInfo
              }
            });
            return { cultivators: [], level: slot.level }; // Clear
all cultivators
          });
          return { ...side, slots: clearedSlots };
        });
        return { ...cart, sides: clearedSides };
      }
      return cart;
    });

    setCarts(newCarts);
    // lastMovedInfo is updated within the loop above, so just set it
    setLastMovedInfo({ ...lastMovedInfo });
    addStateToHistoryAndSave(newCarts, { ...lastMovedInfo });
  };

  // Undo function
  const handleUndo = () => {
    if (historyIndex > 0) {
      const newIndex = historyIndex - 1;
      const prevState = history[newIndex];
      setCarts(prevState.carts);
      setLastMovedInfo(prevState.lastMovedInfo);
      setHistoryIndex(newIndex);
      saveStockroomLayout(prevState.carts, prevState.lastMovedInfo);
// Save undo state
    }
  };

  // Redo function
  const handleRedo = () => {
    if (historyIndex < history.length - 1) {
      const newIndex = historyIndex + 1;
      const nextState = history[newIndex];
      setCarts(nextState.carts);
      setLastMovedInfo(nextState.lastMovedInfo);
      setHistoryIndex(newIndex);
      saveStockroomLayout(nextState.carts, nextState.lastMovedInfo);
// Save redo state
    }
  };
```

```
  if (!isAuthReady) {
    return (
      <div className="flex items-center justify-center min-h-screen
bg-gray-900 text-white">
        Loading authentication...
      </div>
    );
  }

  return (
    <div className="flex flex-col min-h-screen bg-gray-900 text-white
font-inter">
      {/* App Header at the very top */}
      <div className="w-full absolute top-0 left-0 right-0 p-4 flex
flex-col items-center justify-center z-10">
        <h1 className="text-3xl uppercase font-semibold p-2 border-2
border-gray-600 rounded-md text-white bg-gray-800 flex items-center
gap-2 w-full max-w-xs">
          <span className="flex-grow text-center">FlowCart</span>
          <svg width="32" height="32" viewBox="0 0 24 24" fill="none"
xmlns="http://www.w3.org/2000/svg">
            <rect x="0" y="0" width="24" height="4.8" fill="#5BCEFA"/>
            <rect x="0" y="4.8" width="24" height="4.8"
fill="#F5A9B8"/>
            <rect x="0" y="9.6" width="24" height="4.8"
fill="#FFFFFF"/>
            <rect x="0" y="14.4" width="24" height="4.8"
fill="#F5A9B8"/>
            <rect x="0" y="19.2" width="24" height="4.8"
fill="#5BCEFA"/>
          </svg>
        </h1>
      </div>

      {/* "by CC" with continuous lines - now positioned above the
main content area */}
      <div className="w-full flex justify-center mt-24 px-4">
        <div className="flex items-center w-full max-w-xs"> {/* Added
max-w-xs here */}
          <div className="flex-grow border-t-2 border-gray-600"></div>
          <span className="text-gray-400 text-xs mx-2">by CC</span>
          <div className="flex-grow border-t-2 border-gray-600"></div>
        </div>
      </div>

      <div className="flex flex-col md:flex-row flex-1"> {/* Main
content area, takes remaining height */}
        {/* Hidden CC<3 */}
        <div style={{ display: 'none', opacity: 0, height: 0,
overflow: 'hidden' }}>CC<3</div>

        {/* Cultivator Palette */}
```

```
        <div className="w-full md:w-1/4 bg-gray-800 p-4 flex flex-col
items-center md:items-center rounded-lg shadow-lg m-2 md:mt-2">
          <h2 className="text-xl font-bold mb-4 text-center md:text-
left">Cultivators</h2>
          <div className="flex flex-wrap gap-2 w-full justify-center">
            {initialCultivators.map(cultivator => (
              <div
                key={cultivator.id}
                draggable={isEditMode} // Only draggable in edit mode
                onDragStart={(e) => handleDragStartCultivator(e,
cultivator)}
                onDragEnd={handleDragEndCultivator}
                className={`${cultivator.color} hover:opacity-80 text-
white px-4 py-2 rounded-md shadow-md ${isEditMode ? 'cursor-grab
active:cursor-grabbing' : 'cursor-not-allowed opacity-50'} text-center
flex-grow md:flex-grow-0`}
              >
                {cultivator.name}
              </div>
            ))}
          </div>
        </div>

        {/* Stockroom Layout */}
        <div
          ref={stockroomRef}
          className="relative flex-1 bg-gray-800 m-2 rounded-lg
shadow-lg overflow-hidden border-2 border-dashed border-gray-700 p-4
grid grid-cols-2 gap-8 justify-items-center items-start pt-20 md:pt-4"
        >
          {/* Edit/View Mode Controls */}
          <div className="w-full col-span-full flex justify-center
gap-4 mb-4">
            <button
              onClick={() => setIsEditMode(false)}
              className={`px-6 py-3 rounded-md shadow-md font-bold
flex items-center gap-2 ${!isEditMode ? 'bg-blue-600 text-white' :
'bg-gray-600 text-gray-300 hover:bg-blue-500 hover:text-white'}`}
            >
              <Eye size={20} />
              <span>View Mode</span>
            </button>
            <button
              onClick={() => setIsEditMode(true)}
              className={`px-6 py-3 rounded-md shadow-md font-bold
flex items-center gap-2 ${isEditMode ? 'bg-green-600 text-white' :
'bg-gray-600 text-gray-300 hover:bg-green-500 hover:text-white'}`}
            >
              <Pencil size={20} />
              <span>Edit Mode</span>
            </button>
          </div>
```

```
          {/* Undo/Redo Buttons */}
          <div className="w-full col-span-full flex justify-center
gap-2 mb-4">
            <button
              onClick={handleUndo}
              disabled={historyIndex <= 0 || !isEditMode} // Disable
if at start of history or not in edit mode
              className={`p-2 rounded-md shadow-md font-bold flex
items-center justify-center ${historyIndex <= 0 || !isEditMode ? 'bg-
gray-500 cursor-not-allowed' : 'bg-yellow-600 hover:bg-yellow-700'}
text-white`}
              title="Undo last change"
            >
              <RotateCcw size={16} /> {/* Smaller icon */}
            </button>
            <button
              onClick={handleRedo}
              disabled={historyIndex >= history.length - 1 ||
!isEditMode} // Disable if at end of history or not in edit mode
              className={`p-2 rounded-md shadow-md font-bold flex
items-center justify-center ${historyIndex >= history.length - 1 ||
!isEditMode ? 'bg-gray-500 cursor-not-allowed' : 'bg-yellow-600
hover:bg-yellow-700'} text-white`}
              title="Redo last undone change"
            >
              <RotateCw size={16} /> {/* Smaller icon */}
            </button>
          </div>

          {carts.map(cart => (
            <div
              key={cart.id}
              id={cart.id}
              className="w-full max-w-[300px]"
            >
              <Cart
                cart={cart}
                onDropCultivator={handleDropCultivator}
                isDraggingCultivator={isDraggingCultivator}
                userId={userId}
                lastMovedInfo={lastMovedInfo}
                onClearCart={handleClearCart}
                isEditMode={isEditMode} // Pass edit mode status
              />
            </div>
          ))}
        </div>
      </div>

      {/* Status StatusBar - now a fixed footer */}
      <div className="fixed bottom-0 left-0 right-0 bg-gray-700 text-
```

```
white text-center p-2 text-sm z-20">
        {savingStatus && <span className="mr-4">{savingStatus}</span>}
        <span>Your User ID: {userId}</span>
      </div>
    </div>
  );
};

export default App;
```